Docket No.: MP1502
(PATENT)

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

In re Patent Application of:
Joel David Munter, et al.

Application No.: 10/802,586 Confirmation No.: 5149

Filed: March 17, 2004 Art Unit: 2192

For: POWER AND/OR ENERGY OPTIMIZED COMPILE/EXECUTION Examiner: Ben C. Wang

APPEAL BRIEF

Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313-1450

Dear Sir:

This appeal brief is submitted in response to the Final Office action dated November 24, 2009, and from the Notice of Appeal filed February 22, 2010 and the Notice of Panel Decision from Pre-Appeal Brief Review dated June 1, 2010.

(1) REAL PARTY IN INTEREST

The real party in interest is Marvell International, Ltd. by virtue of an Assignment recorded in the Patent Office on November 15, 2006 at Reel 018515, Frame 0817.

(2) RELATED APPEALS AND INTERFERENCES

There are no related appeals or interferences.

(3) STATUS OF CLAIMS

Claims 1-4, 7-11, 13-18, 20-26, and 28-33 are rejected and are appealed. The claims on appeal are provided in the Claims Appendix. The status of the claims is more specifically set forth below.

1. Claims cancelled: 5-6, 12, 19, and 27
2. Claims withdrawn from consideration but not cancelled: None
3. Claims pending: 1-4, 7-11, 13-18, 20-26, and 28-33
4. Claims allowed: None
5. Claims rejected: 1-4, 7-11, 13-18, 20-26, and 28-33
6. Claims appealed: 1-4, 7-11, 13-18, 20-26, and 28-33

(4) STATUS OF AMENDMENTS

No amendments to the claims were made subsequent to the final rejection, and there are no unentered amendments.

(5) SUMMARY OF CLAIMED SUBJECT MATTER

Although specification citations are inserted below in accordance with 37 C.F.R. 1.192(c), these reference numerals and citations are merely examples of where support may be found in the specification for the terms used in this section of the brief. There is no intention to in any way suggest that the terms of the claims are limited to the examples in the specification.

Independent claim 1 is generally directed to a method. The method includes: a) receiving a plurality of non-native instructions in a selected one of a source form and an

intermediate form; (page 5, para. [0015]; page 6, para. [0020]; page 10, para. [0033]; Fig. 1, ref. no. 122; Fig. 3, ref. no. 322); b) determining an initial number of times to interpretively execute the plurality of non-native instructions (page 8, para. [0027]; page 9, para. [0030]; Fig. 4, ref. no. 402); c) interpretively executing the plurality of non-native instructions the initial number of times (page 8, para. [0026-27]; Fig. 4, ref. no. 404); d) monitoring execution of the plurality of non-native instructions to determine when the plurality of non-native instructions have been interpretively executed the initial number of times (page 8, para. [0029]; page 10, para. [0034]); and e) compiling the plurality of non-native instructions to generate object code for the non-native instructions only after interpretively executing the plurality of non-native instructions the initial number of times (page 10, para. [0034]; Fig. 4, ref. no. 406), wherein compiling the plurality of non-native instructions includes replacing an object code segment from the generated object code with an alternative object code segment if the alternative object code segment improves at least a selected one of an execution power level required and an execution energy level required to execute the generated object code in a target execution environment (page 7, para. [0021-22; page 10, para. [0034]; Fig. 2, ref. nos. 204, 206, 208, 210, 212, and 214).

Independent claim 11 is generally directed to a method of operation for an electronic device. The method includes: a) receiving a plurality of non-native instructions (page 5, para. [0015]; page 6, para. [0020]; page 10, para. [0033]; Fig. 1, ref. no. 122; Fig. 3, ref. no. 322); b) determining an initial number of times to interpretively execute the non-native instructions based at least in part on one or more of an expected power level required to perform a compile or an expected energy required to perform the compile (page 8, para. [0027]; page 9, para. [0030]; Fig. 4, ref. no. 402); c) executing the non-native instructions for the initial number of times using an interpreter (page 8, para. [0026-27]; Fig. 4, ref. no. 404); d) monitoring execution of the non-native instructions to determine when the non-native instructions have been executed the initial number of times using the interpreter (page 8, para. [0029]; page 10, para. [0034]); and e) compiling the non-native instructions into object code only after

executing the received non-native instructions for said initial number of times using the interpreter (page 10, para. [0034]; Fig. 4, ref. no. 406).

Independent claim 15 is generally drawn to an article of manufacture. The article includes: a) a computer readable medium (page 4, para. [0014]); and b) a plurality of instructions designed to implement a compiler to (page. 5, para. [0015]); c) determine an initial number of times to interpretively execute a plurality of non-native instructions (page 8, para. [0027]; page 9, para. [0030]; Fig. 4, ref. no. 402); d) interpretively execute the plurality of non-native instructions the initial number of times (page 8, para. [0026-27]; Fig. 4, ref. no. 404); e) monitor execution of the non-native instructions to determine when the non-native instructions have been interpretively executed the initial number of times (page 8, para. [0029]; page 10, para. [0034]); f) compile the non-native instructions[[,]] to generate object code for the non-native instructions only after interpretively executing the plurality of non-native instructions the initial number of times (page 10, para. [0034]; Fig. 4, ref. no. 406); and g) replace a segment of the generated object code with an alternative object code segment if the alternative object code segment improves at least a selected one of an execution power level required and an execution energy required to execute the generated object code in a target execution environment (page 7, para. [0021-22; page 10, para. [0034]; Fig. 2, ref. nos. 204, 206, 208, 210, 212, and 214).

Independent claim 18 is generally drawn to an article of manufacture. The article comprises: a) a computer readable medium (page 4, para. [0014]); b) a plurality of instructions designed to implement a runtime manager equipped to (page. 5, para. [0015]); c) receive a plurality of non-native instructions (page 5, para. [0015]; page 6, para. [0020]; page 10, para. [0033]; Fig. 1, ref. no. 122; Fig. 3, ref. no. 322); d) determine an initial number of times to interpretively execute the non-native instructions, the initial number of times based at least in part on one or more of an expected power level required to perform an average compile or an expected energy required to perform the average compile (page 8, para. [0027]; page 9, para. [0030]; Fig. 4, ref. no. 402); e) execute the non-native instructions for said initial number of times using an interpreter,

(page 8, para. [0026-27]; Fig. 4, ref. no. 404); f) monitor execution of the non-native instructions to determine when the non-native instructions have been interpretively executed the initial number of times (page 8, para. [0029]; page 10, para. [0034]); and g) invoke a compiler to compile the non-native instructions into object code only after executing the received non-native instructions for said initial number of times using the interpreter (page 10, para. [0034]; Fig. 4, ref. no. 406).

Independent claim 22 is generally drawn to a system. The system comprises: a) a storage medium having stored therein a plurality of instructions implementing a compiler to (page 4, para. [0014]); b) receive a plurality of non-native instructions in a selected one of a source form and an intermediate form (page 5, para. [0015]; page 6, para. [0020]; page 10, para. [0033]; Fig. 1, ref. no. 122; Fig. 3, ref. no. 322); c) determine an initial number of times to interpretively execute the plurality of non-native instructions (page 8, para. [0027]; page 9, para. [0030]; Fig. 4, ref. no. 402); d) interpretively execute the plurality of non-native instructions the initial number of times (page 8, para. [0026-27]; Fig. 4, ref. no. 404); e) monitor execution of the non-native instructions to determine when the non-native instructions have been interpretively executed the initial number of times (page 8, para. [0029]; page 10, para. [0034]); f) compile the non-native instructions to generate object code for the non-native instructions only after interpretively executing the plurality of non-native instructions the initial number of times (page 10, para. [0034]; Fig. 4, ref. no. 406); g) replace a segment of the generated object code with an alternative object code segment if the alternative object code segment improves at least a selected one of an execution power level required and an execution energy required to execute the generated object code in a target execution environment (page 7, para. [0021-22; page 10, para. [0034]; Fig. 2, ref. nos. 204, 206, 208, 210, 212, and 214); and h) a processor coupled to the storage medium to execute the instructions implementing the compiler (page 4, para. [0014]; Fig. 1, ref. no. 102).

Independent claim 26 is generally drawn to a system. The system comprises: a) a communication interface to receive a plurality of non-native instructions (page 5, para.

[0015]; page 6, para. [0020]; page 10, para. [0033]; Fig. 1, ref. no. 122; Fig. 3, ref. no. 322); b) a storage medium coupled to the communication interface, and having stored therein a plurality of instructions designed to implement a runtime manager equipped to (page 4, para. [0014]); c) determine an initial number of times to interpretively execute the non-native instructions, the initial number of times based at leas103. t in part on one or more of an expected power level required to perform an average compile or an expected energy required to perform the average compile (page 8, para. [0027]; page 9, para. [0030]; Fig. 4, ref. no. 402); d) execute the received non-native instructions for the initial number of times using an interpreter (page 8, para. [0026-27]; Fig. 4, ref. no. 404); e) monitor execution of the non-native instructions to determine when the non-native instructions have been interpretively executed the initial number of times (page 8, para. [0029]; page 10, para. [0034]); f) invoke a compiler to compile the non-native instructions into object code only after executing the received non-native instructions for said initial number of times using the interpreter (page 10, para. [0034]; Fig. 4, ref. no. 406); and g) a processor coupled to the storage medium to execute the instructions implementing the runtime manager (page 4, para. [0014]; Fig. 1, ref. no. 102).

(6) GROUNDS OF REJECTION TO BE REVIEWED ON APPEAL

A. Whether claims 1-4, 7-11, 13-18, 20-26, and 28-33 are unpatentable under 35 U.S.C. §103(a) U.S. Pat. Pub. No. 20040010785, hereinafter "Chauvel" in view of U.S. Pat. Pub. No. 20050114850, hereinafter "Chheda".

(7) ARGUMENT

A. <u>The Examiner Has Failed to Establish that Claims 1-4, 7-11, 13-18, 20-26, and 28-33 Are Unpatentable Over Chauvel in View of Chheda Under 35 U.S.C. §103(a).</u>

1. Overview

Claims 1-4, 7-11, 13-18, 20-26, and 28-33 stand rejected under 35 U.S.C. §103(a) as allegedly being unpatentable over Chauvel in view of Chheda. Applicants respectfully submit that the Examiner has failed to establish a *prima facie* case that claims 1-4, 7-11, 13-18, 20-26, and 28-33 are obvious over Chauvel in view of Chheda.

2. Summary of Independent Claims, Chauvel, and Chheda

As discussed below, neither Chauvel nor Chheda discloses or suggests at least, after receiving a plurality of non-native instructions:

a) determining an initial number of times to interpretively execute the plurality of non-native instruction (claims 1, 11, 15, 18, 22, 26)

b) monitoring execution of the plurality of non-native instructions to determine when the plurality of non-native instructions have been interpretively executed the initial number of times or executed the initial number of times using the interpreter (claims 1, 11, 15, 18, 22 and 26)

c) compiling the plurality of non-native instructions to generate object code for the plurality of non-native instructions only after interpretively executing the plurality of non-native instructions the initial number of times (claims 1, 11, 15, 18, 22 and 26).

Chauvel describes profiling or estimating performance characteristics of an application during execution by comparing an application profile against a virtual machine profile. The byte-code based profile of the application indicates how many times the application executes particular operations or methods.[1] A virtual machine

[1] "By independently generating the application profile, based on the number of times operations are executed in the application, and the virtual machine profile, based on actual hardware response on the target device, an accurate estimation of a performance criteria such as average time, maximum time , or energy consumption, for the application can be provided." Chauvel, Paragraph [0013], lines 2-8 (emphasis added).

profile then relates a performance characteristic to the operations or methods and generates an aggregate value for the performance characteristic. An API implementing a modified interpreter loop and other modifications to a Java Virtual Machine (JVM) generates the virtual machine profile. The virtual machine profile provides information indicative of a performance characteristic (*i.e.*, execution time, energy consumption, or power level) of the device's underlying hardware for each particular application operation. The application profile and the virtual machine profile are then combined to generate a performance estimate to aid a programmer in optimizing the application or to aid in efficient operation of the device.[2] The generated performance estimate may then be used for scheduling operations. Therefore, by comparing an application profile that is based on a number of times an operation appears in a code segment and a virtual machine profile, Chauvel generally describes generating an application performance estimate so that particular application operations may be re-scheduled before execution to optimize the performance of the device.

Chheda generally describes analyzing and transforming program instructions in an executable form by re-compiling the instructions to, *inter alia*, reduce energy consumption in a processor.[3] Chheda generally describes executable-level re-compilation to perform analysis and modifications on an executable file. In particular, Chheda describes scanning an executable file and creating a version of the file based on an energy-focused Binary Intermediate Format (BIF). The BIF file is an abstract representation of the executable that organizes the file into an inter-related structure to help perform further analysis.[4] For example, the BIF file permits the identification of

---

[2] "By multiplying the number of times each operation is used in execution of the application (as specified in the application profile 10) by the energy/time consumed by the operation (as specified in the JVM profile 14) and summing the results for all such operations, a very accurate estimation of the time energy used by the application can be generated." Chauvel, Paragraph [0028] lines 3-9 (emphasis added).

[3] "The executable-level modification or executable re-compilation, and compiler-architecture interaction based processor framework described herein addresses the foregoing need to reduce energy consumption in a practical and flexible manner." Chheda, para. [0032], lines 1-5 (emphasis added).

program dependencies, loops, etc.[5] Once identified, a "criticality" weight is assigned to the various, identified portions of the BIF file. Portions of the BIF file that consume more energy (*e.g.,* loops) receive a higher weight than those portions that consume less energy.[6] Those portions are then analyzed to remove redundant micro-operations or scheduled to reduce the amount of energy consumed by the executable.[7] In sum, Chheda generally describes analyzing a received executable file to identify critical portions of code for analysis and removal or re-scheduling of redundant micro-operations.

3. Analysis

Claim 1, 11, 15, 18, 22, and 26 are generally directed to a method, article of manufacture, or system for optimizing power/energy consumed by a computing device while compiling or executing non-native instructions in source or intermediate form. The recited method, article of manufacture, or system generally recites determining how many times to execute, interpretively, non-native instructions an initial number of times, and executing the instructions. The execution is monitored to determine when the instructions have been executed the initial number of times. The instructions are then compiled to generate object code from the received non-native instructions only after the non-native instructions have been interpretively executed an initial number of times.

---

[4] "BIF is an abstract representation that facilitates easy retargetability of the analysis and transformation techniques to different instruction sets. It provides necessary information organized in inter-related structures to help perform sophisticated program analyses and transformations." Chheda, para. [0077].

[5] *See* Chheda, paragraphs [0076-0082].

[6] *See* Chheda, para. [0083-0084].

[7] "Many of these micro-operations are not necessary if there is related information extracted in the compiler that provides it, or if there is program information that enables a way to replace these micro-operations with more energy-efficient but equivalent ones. " Chheda, para. [0045], lines 5-9; *also* "Furthermore, the method may include inserting a more energy efficient instruction replacing an existing instruction and may include inserting control bits to control hardware structures." Chheda, para. [0050], lines 1-4.

The teachings of Chauvel, either alone or in combination with Chheda, do not teach or suggest the recitations of claims 1, 11, 15, 18, 22, or 26 discussed above and as alleged in the November 24, 2009 Final Office Action. In an interview with the Examiner on December 9, 2008 regarding the non-final office action dated October 15, 2008, the parties discussed the rejection of claims 1, 11, 15, 18, 22, and 26 and, in particular, the alleged teachings of Chauvel. At the conclusion of this interview, the examiner agreed that Chauvel did not teach or suggest the recitations of the pending claims. In particular, the examiner agreed that Chauvel generally teaches determining how many times a code segment is executed in a given code segment rather than determining how many times to execute, interpretively, non-native instructions an initial number of times as generally recited in claims 11, 18, and 26 (and as added by subsequent amendment to claims 1, 15, and 22). In the office action immediately following this interview, the Examiner did not allege that the teachings of Chauvel taught or suggested any of claims 1, 11, 15, 18, 22, and 26 recitations. The Examiner instead relies solely on the alleged teachings of Chheda and Chen (Energy-Aware Compilation and Execution in Java-Enabled Mobile Devices, 2003 IEEE, pp. 1-8).

Despite the agreements reached by Applicants' representative and the Examiner regarding the alleged teachings of Chauvel, the November 24, 2009 Final Office Action again cites Chauvel as allegedly teaching or suggesting the "determining..." recitation of claims 1, 11, 15, 18, 22, and 26. Applicants submit that the November 24, 2009 Final Office Action continues to misconstrue the teachings of Chauvel with respect to the "determining..." recitations. As discussed above, Chauvel generally describes creating a runtime profile for the execution of compiled code that includes, among other things, relating power consumption to the number of times an individual operation appears within the code.

As described on page 15 of the October 15, 2008 Office Action and on page 4 of the November 24, 2009 Final Office Action (applied to the "determining..." claim recitations), Chauvel describes:

> an application profile that specifies a number of executions of a plurality of operations used in the specified portion of the application [and] the application profile based on the number of times operations are executed in the application (emphasis added).[8]

Rather than merely counting the number of executions of an operation within a set of instructions, as generally disclosed by Chauvel, independent claims 1, 11, 15, 18, 22, and 26 generally recite determining how many times to execute, interpretively, non-native instructions an initial number of times. An application profile based on counting how many times an operation appears within an application as generally disclosed by Chauvel does not teach or suggest determining an initial number of times to interpretively execute non-native instructions as generally recited in the independent claims.

Additionally, Chauvel does not disclose or suggest monitoring execution of the plurality of non-native instructions to determine when the plurality of non-native instructions have been interpretively executed the initial number of times as recited in claims 1, 11, 15, 18, 22, and 26. The Office Action cites paragraph [0013] of Chauvel as allegedly disclosing this element. But paragraph [0013] merely states that an application profile is generated "based on the number of times operations are executed in the application." As explained elsewhere in Chauvel, this statement merely refers to determining how many times an operation is performed during execution of the application.[9] Determining how many times an operation is performed is not the same as

---

[8] *See* Chauvel, paragraph [0012], lines 5-7 and paragraph [0013], lines 3-5.

[9] *See* paragraph [0024], lines 3-5: "The application profile 10 is a byte-code based profile which indicates how many times each operation (such as a byte-code or a native method) is used during execution of the application 12, or in specified parts of the application 12"; *also* paragraph [0027], lines 1-3: "The application profile 10 specifies the number of times each possible operation is executed during execution of the execution of application 12."; *also* paragraph [0029]: "At this point, the execution of each operation in the application 12 increments a counter associated with the particular operation. Each executed operation is counted until an off command (described below) is received." *also* paragraph [0035], lines 1-4: "The application profile 10 documents the number of times each standard JAVA byte-code (with and without wide) is executed and the number of times each native method of the JVM is executed" (emphasis added).

determining when instructions have been interpretively executed an already determined initial number of times as generally recited in the independent claims.

The Office Action also cited paragraph [0031] of Chauvel as allegedly disclosing "monitoring... to determine when the plurality of non-native instructions have been interpretively executed the initial number of times" generally recited by claims 1, 11, 15, 18, 22, and 26. But, paragraph [0031] generally refers to analysis procedures for wide byte-codes, not whether instructions have been interpretively executed an initial number of times.[10] An analysis procedure for wide byte-codes is not the same as determining when instructions have been interpretively executed a determined initial number of times as generally recited by claims 1, 11, 15, 18, 22, and 26. Thus, Chauvel does not teach or suggest the recitations of claims 1, 11, 15, 18, 22, and 26 and the Examiner's continued misunderstanding of Chauvel cannot be a basis for the current rejections of these independent claims.

On page 5, the November 24, 2009 Final Office Action admits that Chauvel does not explicitly disclose the "compiling..." element of claim 1 and cites Chheda for disclosing the same. However, the Office Action also cites the Abstract of Chauvel as disclosing a profiling system to indicate the number of operation executions in an application and a virtual machine profile that indicates the time/energy consumed by the operation on a particular platform. The Office Action offers no explicit reasoning for citing this portion of Chauvel with reference to the "compiling..." element of claim 1 or claims 11, 15, 18, 22, and 26. Nevertheless, like the portions of Chauvel discussed above, this cited portion of Chauvel merely describes determining how many times an operation is performed when an application is executed and does not disclose or suggest the "initial number of times..." element recited by claims 1, 11, 15, 18, 22, and 26.

Chheda also does not disclose or suggest the above-discussed elements that Chauvel does not describe and, in particular, does not disclose or suggest "compiling

[10] *See* Chauvel, paragraph [0031].

the plurality of non-native instructions to generate object code for the plurality of non-native instructions only after interpretively executing the plurality of non-native instructions the initial number of times" as recited by independent claims 1, 11, 15, 18, 22, and 26. Rather, Chheda generally describes executable level re-compilation approach and identification and removal or rescheduling of redundant micro-operations, as described above. Executable level re-compilation and removal or rescheduling of redundant micro-operations as described by Chheda discloses nothing regarding compilation of non-native instructions only after interpretively executing the non-native instructions the determined initial number of times. Simply put, the executable file described by Chheda cannot teach or suggest the non-native instructions recited by the independent claims. Further, while Chheda generally describes removing or rescheduling redundant micro-operations, such description does not teach or suggest such actions only after interpretively executing the plurality of non-native instructions the initial number of times as recited by claims 1, 11, 15, 18, 22, and 26. Thus, Chheda fails to cure the deficiencies of Chauvel, and the Examiner's continued misunderstanding of Chheda cannot be a basis for the current rejections of the independent claims.

At least for the reasons discussed above, claims 1, 11, 15, 18, 22, and 26 are patentable over Chauvel in view of Chheda. The law on obviousness is clear that all the elements from a claim must be present in the combined prior art references. Chauvel in view of Chheda does not disclose or suggest all the elements from claims 1, 11, 15, 18, 22, and 26. Accordingly, claims 1, 11, 15, 18, 22, and 26 are patentable over Chauvel in view of Chheda.

Claims 2-4, 7-10, 13, 14, 16, 17, 20, 21, 23-25, and 28-33 stand rejected under 35 U.S.C. §103(a) as allegedly being unpatentable over Chauvel in view of Chheda. Applicants respectfully submit that the Examiner has failed to establish that claims 2-4, 7-10, 13, 14, 16, 17, 20, 21, 23-25, and 28-33 are unpatentable over Chauvel in view of Chheda. Accordingly, the rejection of these claims represents an error and should be overturned.

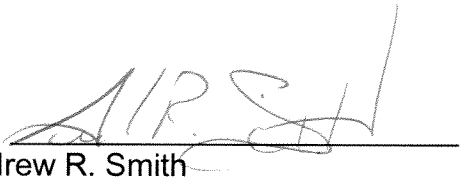
Applicants note that if an independent claim is nonobvious under 35 U.S.C. §103, then any claim depending therefrom is nonobvious. Because all of claims 2-4, 7-10, 13, 14, 16, 17, 20, 21, 23-25, and 28-33 depend from independent claims that are nonobvious, claims 2-4, 7-10, 13, 14, 16, 17, 20, 21, 23-25, and 28-33 are also nonobvious.

4. Conclusion

At least for the foregoing reasons, Applicants respectfully request that the rejections of claims 1-4, 7-11, 13-18, 20-26, and 28-33 be reversed.

Respectfully submitted,

Date: June 29, 2010

By: 

Andrew R. Smith
Registration No. 62,162
Attorneys for Applicants
MARSHALL, GERSTEIN & BORUN
233 South Wacker Drive
Chicago, Illinois 60606-6402
(312) 474-6300

CLAIMS APPENDIX

1. A method comprising:

receiving a plurality of non-native instructions in a selected one of a source form and an intermediate form;

determining an initial number of times to interpretively execute the plurality of non-native instructions;

interpretively executing the plurality of non-native instructions the initial number of times;

monitoring execution of the plurality of non-native instructions to determine when the plurality of non-native instructions have been interpretively executed the initial number of times;

compiling the plurality of non-native instructions to generate object code for the non-native instructions only after interpretively executing the plurality of non-native instructions the initial number of times, wherein compiling the plurality of non-native instructions includes replacing an object code segment from the generated object code with an alternative object code segment if the alternative object code segment improves at least a selected one of an execution power level required and an execution energy level required to execute the generated object code in a target execution environment.

2. The method of claim 1, wherein said receiving comprises receiving the non-native instructions in a byte code form.

3. The method of claim 1, wherein said compiling comprises analyzing the object code segment for execution power level requirement, and determining whether an alternative object code segment with lower execution power level requirement is available.

4. The method of claim 1, wherein said compiling comprises analyzing the object code segment for execution energy consumption, and determining whether an alternative object code segment with lower execution energy consumption is available.

5-6. (Canceled)

7. The method of claim 6, wherein the method further comprises

monitoring said compiling for power level required to perform compilation;

updating a current understanding of power level required for compilation; and

updating the initial number of times the plurality of non-native instructions are to be interpretively executed before compiling the received non-native instructions, if said monitoring observes a power level required for compilation to be different from the current understanding.

8. The method of claim 6, wherein the method further comprises

monitoring said compiling for amount of energy required to perform an average compilation;

updating a current understanding of amount of energy required for an average compilation; and

updating the initial number of times the plurality of non-native instructions are to be interpretively executed before compiling the received non-native instructions, if said monitoring observes an amount of energy required for compilation to be different from the current understanding.

9. The method of claim 1, wherein the generated object code comprises a plurality of native instructions, and the method further comprises

monitoring execution of the generated object code for power level required to execute the native instructions; and

updating power level requirements of selected ones of the native instructions if said monitoring observes power level requirements for the selected ones of the native instructions to be different from current understandings of the power level requirements of the selected ones of the native instructions.

10. The method of claim 1, wherein the generated object code comprises a plurality of native instructions, and the method further comprises

monitoring execution of the generated object code for amount of energy required to execute the native instructions; and

updating energy requirements of selected ones of the native instructions if said monitoring observes energy requirements for the selected ones of the native

instructions to be different from current understandings of the energy requirements of the selected ones of the native instructions.

11. In an electronic device, a method of operation, comprising:

receiving a plurality of non-native instructions;

determining an initial number of times to interpretively execute the non-native instructions based at least in part on one or more of an expected power level required to perform a compile or an expected energy required to perform the compile;

executing the non-native instructions for the initial number of times using an interpreter;

monitoring execution of the non-native instructions to determine when the non-native instructions have been executed the initial number of times using the interpreter; and

compiling the non-native instructions into object code only after executing the received non-native instructions for said initial number of times using the interpreter.

12. (Canceled)

13. The method of claim 11, wherein the method further comprises monitoring said compiling for a compilation requirement employed in determining the initial number of times the received non-native instructions are to be executed using the interpreter before compiling; and

updating a current understanding of the compilation requirement if said monitoring observes the compilation requirement to be different from the current understanding.

14. The method of claim 11, wherein the generated object code comprises a plurality of native instructions, and the method further comprises monitoring execution of the generated object code for execution requirements of the native instructions; and

updating execution requirements of selected ones of the native instructions if said monitoring observes execution requirements for the selected ones of the native instructions to be different from current understandings of the execution requirements of the selected ones of the native instructions.

15. An article of manufacture comprising:

a computer readable medium; and

a plurality of instructions designed to implement a compiler to:

determine an initial number of times to interpretively execute a plurality of non-native instructions;

interpretively execute the plurality of non-native instructions the initial number of times;

monitor execution of the non-native instructions to determine when the non-native instructions have been interpretively executed the initial number of times;

compile the non-native instructions to generate object code for the non-native instructions only after interpretively executing the plurality of non-native instructions the initial number of times, and

replace a segment of the generated object code with an alternative object code segment if the alternative object code segment improves at least a selected one of an execution power level required and an execution energy required to execute the generated object code in a target execution environment.

16. The article of claim 15, wherein said compiler analyzes the object code segment for execution power level requirement, and determines whether the alternative object code segment with lower execution power level requirement is available.

17. The article of claim 15, wherein said compiler analyzes the object code segment for execution energy consumption, and determines whether the alternative object code segment with lower execution energy consumption is available.

18. An article of manufacture comprising:

a computer readable medium; and

a plurality of instructions designed to implement a runtime manager equipped to:

receive a plurality of non-native instructions,

determine an initial number of times to interpretively execute the non-native instructions, the initial number of times based at least in part on one or more of an expected power level required to perform an average compile or an expected energy required to perform the average compile,

execute the non-native instructions for said initial number of times using an interpreter,

monitor execution of the non-native instructions to determine when the non-native instructions have been interpretively executed the initial number of times, and

invoke a compiler to compile the non-native instructions into object code only after executing the received non-native instructions for said initial number of times using the interpreter.

19. (Canceled)

20. The article of claim 18, wherein the runtime manager is further equipped to monitor said compiling for a compilation requirement employed in determining the initial number of times received non-native instructions are to be executed before compiling; and

update a current understanding of the compilation requirement if said monitoring observes the compilation requirement to be different from the current understanding.

21. The article of claim 18, wherein the generated object code comprises a plurality of native instructions, and the runtime manager is further equipped to monitor execution of the generated object code for execution requirements of the native instructions; and

update execution requirements of selected ones of the native instructions if said monitoring observes execution requirements for the selected ones of the native instructions to be different from current understandings of the execution requirements of the selected ones of the native instructions.

22. A system, comprising:

a storage medium having stored therein a plurality of instructions implementing a compiler to:

receive a plurality of non-native instructions in a selected one of a source form and an intermediate form,

determine an initial number of times to interpretively execute the plurality of non-native instructions;

interpretively execute the plurality of non-native instructions the initial number of times,

monitor execution of the non-native instructions to determine when the non-native instructions have been interpretively executed the initial number of times,

compile the non-native instructions to generate object code for the non-native instructions only after interpretively executing the plurality of non-native instructions the initial number of times, and

replace a segment of the generated object code with an alternative object code segment if the alternative object code segment improves at least a selected one of an execution power level required and an execution energy required to execute the generated object code in a target execution environment; and

a processor coupled to the storage medium to execute the instructions implementing the compiler.

23. The system of claim 22, wherein said compiler analyzes the object code segment for execution power level requirement, and determines whether an alternative object code segment with lower execution power level requirement is available.

24. The system of claim 22, wherein said compiler analyzes the object code segment for execution energy consumption, and determines whether an alternative object code segment with lower execution energy consumption is available.

25. The system of claim 22, wherein the apparatus further comprises a wireless communication interface to receive the non-native instructions.

26. A system, comprising:

a communication interface to receive a plurality of non-native instructions;

a storage medium coupled to the communication interface, and having stored therein a plurality of instructions designed to implement a runtime manager equipped to:

determine an initial number of times to interpretively execute the non-native instructions, the initial number of times based at least in part on one or more of an expected power level required to perform an average compile or an expected energy required to perform the average compile,

execute the received non-native instructions for the initial number of times using an interpreter,

monitor execution of the non-native instructions to determine when the non-native instructions have been interpretively executed the initial number of times, and

invoke a compiler to compile the non-native instructions into object code only after executing the received non-native instructions for said initial number of times using the interpreter; and

a processor coupled to the storage medium to execute the instructions implementing the runtime manager.

27. (Canceled)

28. The system of claim 26, wherein the runtime manager is further equipped to monitor said compiling for a compilation requirement employed in

determining the initial number of times received non-native instructions are to be executed using the interpreter before compiling; and

update a current understanding of the compilation requirement if said monitoring observes the compilation requirement to be different from the current understanding.

29. The system of claim 26, wherein the generated object code comprises a plurality of native instructions, and the runtime manager is further equipped to

monitor execution of the generated object code for execution requirements of the native instructions; and

update execution requirements of selected ones of the native instructions if said monitoring observes execution requirements for the selected ones of the native instructions to be different from current understandings of the execution requirements of the selected ones of the native instructions.

30. The system of claim 26, wherein the communication interface is a wireless communication interface.

31. The method of claim 11, wherein determining the initial number of times to interpretively execute the non-native instructions is based at least in part on a size of the non-native instructions.

32. The article of claim 18, wherein the runtime manager is further equipped to determine the initial number of times to interpretively execute the non-native instructions based at least in part on a size of the non-native instructions.

33. The system of claim 26, wherein the runtime manager is further equipped to determine the initial number of times to interpretively execute the non-native instructions based at least in part on a size of the non-native instructions.

EVIDENCE APPENDIX

None.

RELATED PROCEEDINGS APPENDIX

None.